ML2_Final (after A100

April 26, 2025

1 Samantic Clustering of Wikipedia Articles with unsupervised learning

1.1 Problem

I thought I wanted to cluster some Semantic data, but how would I do It? I don't want to label thousands of orticles, also mabe the labels I set are not good clusters. I want this dome automaticly with unsupervised learning

1.2 Getting the Data, performing EDA, and Data cleaning

I got the Data from Wikidumps, to get the Data passable I needed to edit the Wikidump and export it to json. I did this with WikiExtractor wich is an open source opten for that (even though it was hart to get to function because of its age)

Bercause it Split the entire dump (20GB) into small 1MB chunks, first I needed to combine the again

I have to preclean the Data because I want to transform it with semantic embedding (extract the meaning out of the text) for that I have to deate the stopwords and optimaly reduce the number of words / deleate the pages who have no words, and so on. My dataset features are Title ID and Text. Title and Text are strings ID is an integer. In the futer I will also have Embedding as a numpy arry and more. (the more will mostly be arrays that I saved to work on them later)

[ ]:
```
import os
import json
from pathlib import Path
import pandas as pd
from tqdm import tqdm


data_root = Path("PATH")
docs = []
for file in tqdm(list(data_root.rglob("*")), desc="Reading files"):
    if not file.is_file():
        continue
    with open(file, "r", encoding="utf-8") as f:
        for line in f:
            line = line.strip()
```

```
            if not line:
                continue
            try:
                obj = json.loads(line)
                docs.append({
                    "id":    obj.get("id"),
                    "title": obj.get("title", ""),
                    "text":  obj.get("text", "")
                })
            except json.JSONDecodeError:
                continue



df = pd.DataFrame(docs)
print(f"Loaded {len(df)} documents")
```

1.2.1 Basic EDA

First I counted the number of words and charecters per Artice, and displayed them in a graph I also looked at the distribution of the number of words/ chars from the articles

[ ]:
```
df['char_count'] = df['text'].apply(len)
df['word_count'] = df['text'].apply(lambda x: len(x.split()))

print(df[['char_count', 'word_count']].describe())

df['word_count'].hist(bins=50)
```

```
         char_count    word_count
count  4.893061e+06  4.893061e+06
mean   1.512650e+03  2.105370e+02
std    4.025713e+03  5.534175e+02
min    0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00
50%    3.250000e+02  4.600000e+01
75%    1.758000e+03  2.480000e+02
max    5.916920e+05  8.343700e+04
```

[ ]: `<Axes: >`

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import re


%matplotlib inline
plt.style.use("ggplot")


df_words = df[['word_count']].copy()

fig, ax = plt.subplots(figsize=(10, 6))
bins = np.arange(0, 10_001, 500)
ax.hist(df_words['word_count'], bins=bins, edgecolor="black")
ax.set(
    title="Distribution of article lengths (words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
```

```python
plt.show()

fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(df_words.loc[df_words['word_count'] <= 2000, 'word_count'],
        bins=40, edgecolor="black")
ax.set(
    title="Distribution of shorter articles (  2 000 words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()


fig, ax = plt.subplots(figsize=(8, 5))
ax.boxplot(df_words['word_count'], vert=False, showfliers=False)
ax.set(
    title="Boxplot of article word counts",
    xlabel="Words per article"
)
plt.show()


df_len = df[['word_count', 'char_count']].copy()

fig, ax = plt.subplots(figsize=(8, 6))
ax.scatter(df_len['word_count'], df_len['char_count'],
           alpha=0.05, s=5)
ax.set(
    title="Characters vs. Words per article",
    xlabel="Words per article",
    ylabel="Characters per article"
)
plt.show()


fig, ax = plt.subplots(figsize=(8, 6))
counts, edges = np.histogram(df_words['word_count'], bins=100)
cdf = np.cumsum(counts) / counts.sum()
ax.plot(edges[1:], cdf)
ax.set(
    title="CDF - proportion of articles up to length X",
    xlabel="Words per article",
    ylabel="Cumulative proportion"
)
plt.show()
```

```
top20 = df.nlargest(20, 'word_count')[['title', 'word_count']].
 ↪reset_index(drop=True)
display(top20)


def simple_tokens(text: str):
    """Lower-cases & grabs alphanum tokens."""
    return re.findall(r'\b\w+\b', text.lower())

sample_size = 50_000
sample = df.sample(min(sample_size, len(df)), random_state=42)

tok_counter = Counter()
for doc_text in sample['text']:
    tok_counter.update(simple_tokens(doc_text))

top_words = pd.DataFrame(tok_counter.most_common(30),
                         columns=['token', 'freq'])

fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(top_words['token'][::-1], top_words['freq'][::-1])
ax.set(
    title="Top 30 tokens in a 50 k-article sample",
    xlabel="Frequency"
)
plt.show()
```

	title	word_count
0	Chronik der COVID-19-Pandemie in den Vereinigt…	83437
1	Chronik der COVID-19-Pandemie in den Vereinigt…	79832
2	Liste von Filmen mit homosexuellem Inhalt	78666
3	Giacomo Casanova	73974
4	Frankfurt am Main in der Literatur	65648
5	Chronik der COVID-19-Pandemie in den Vereinigt…	63418
6	Bauwerke in Bockenheim	55837
7	Belle Époque	55018
8	Figuren im Star-Trek-Universum	52927
9	Geschichte Osttimors	49132
10	Inhalt und Interpretation der Unendlichen Gesc…	48020
11	Russland	45928
12	Park Klein-Glienicke	41228
13	Geschichte des Saarlandes	39386
14	Scuderia Ferrari	39117
15	Liste der Stolpersteine in Tübingen Innenstadt	38716
16	Die Schlümpfe (Comic-Geschichten)	37116
17	Spätantike	36916
18	Arte 360°-Reportage	36761

```
19                   Figuren im Lied von Eis und Feuer       36098
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#  Build throw-away subsets
df_0_500   = df.loc[df['word_count']       <=  500].copy()
df_500_1k  = df.loc[(df['word_count'] > 500) &
                    (df['word_count'] <= 1000)].copy()
df_0_1k    = df.loc[df['word_count']       <= 1000].copy()


fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharey=True)

# 0-500 words
axes[0].hist(df_0_500['word_count'],
             bins=np.arange(0, 501, 25), edgecolor="black")
axes[0].set(title="Articles 0 - 500 words",
            xlabel="Words per article", ylabel="Number of articles")

# 500-1 000 words
axes[1].hist(df_500_1k['word_count'],
             bins=np.arange(500, 1001, 25), edgecolor="black")
```

```
axes[1].set(title="Articles 500 - 1 000 words",
            xlabel="Words per article")

plt.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(8, 5))
ax.boxplot(
    [df_0_500['word_count'],
     df_500_1k['word_count'],
     df_0_1k['word_count']],
    labels=["0-500", "500-1 000", "0-1 000"],
    vert=False,
    showfliers=False
)
ax.set(title="Word-count distribution by range",
       xlabel="Words per article")
plt.show()


fig, ax = plt.subplots(figsize=(8, 6))
counts, edges = np.histogram(df_0_1k['word_count'], bins=100)
ax.plot(edges[1:], np.cumsum(counts) / counts.sum())
ax.set(title="CDF for articles  1 000 words",
       xlabel="Words per article",
       ylabel="Cumulative proportion")
plt.show()


bands = pd.Series({
    "0-500"      : len(df_0_500),
    "500-1 000"  : len(df_500_1k),
    "1 000-∞"    : len(df) - len(df_0_1k)
})
fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(bands.index, bands.values)
ax.set(title="Article-count by length band",
       ylabel="Number of articles")
for i, v in enumerate(bands.values):
    ax.text(i, v, f"{v:,}", ha="center", va="bottom")
plt.show()
```

```
/var/folders/yx/8v6g52y10p9gxhgxm0dvr2vr0000gn/T/ipykernel_727/1900998789.py:41:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been
renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be
dropped in 3.11.
  ax.boxplot(
```

1.3 EDA First results

If you look at the graphs you will see that there is a high increase in the number of articles the shorter they are, this seems logical. There also are some major outlires when it comes to article lenghth. We will take a menal note of that and process that later

The distiribution indicates that there are a number of articles where the number of wordas is 0 I cant group them semanticly when I havce no information.

[ ]:
```
num_zero_word = (df['word_count'] == 0).sum()
total         = len(df)

print(f"{num_zero_word:,} of {total:,} pages have 0 words "
      f"({num_zero_word/total:.2%} of the corpus).")
```

```
1,914,318 of 4,893,061 pages have 0 words (39.12% of the corpus).
```

1.3.1 Fist Data Cleaning

as you can see 1.9 Million articles have no words. We can drop them as they are not computable for my clustering

[ ]:
```
before = len(df)
df_clean = df.loc[df['word_count'] > 0].reset_index(drop=True)
after = len(df_clean)

print(f"Dropped {before - after:,} zero-word pages "
      f"({(before - after) / before:.2%} of the corpus).")
print(f"Remaining pages: {after:,}")
df = df_clean
```

```
Dropped 1,914,318 zero-word pages (39.12% of the corpus).
Remaining pages: 2,978,743
```

1.3.2 EDA after loosing 1.9 Million articles

because the number of articles that were deleated was so large I wanted to perform the first EDA step again but now without the articles where the number of words is 0

[ ]:
```
df['char_count'] = df['text'].apply(len)
df['word_count'] = df['text'].apply(lambda x: len(x.split()))

print(df[['char_count', 'word_count']].describe())

df['word_count'].hist(bins=50)
```

```
         char_count    word_count
count  2.978743e+06  2.978743e+06
```

```
mean   1.757382e+03  2.028616e+02
std    3.437207e+03  3.863599e+02
min    0.000000e+00  0.000000e+00
25%    4.080000e+02  4.900000e+01
50%    9.950000e+02  1.170000e+02
75%    1.959000e+03  2.290000e+02
max    4.099510e+05  4.914400e+04
```

[ ]:

```
<Axes: >
```

[ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import re

%matplotlib inline
plt.style.use("ggplot")

df_words = df[['word_count']].copy()
```

```
fig, ax = plt.subplots(figsize=(10, 6))
bins = np.arange(0, 10_001, 500)
ax.hist(df_words['word_count'], bins=bins, edgecolor="black")
ax.set(
    title="Distribution of article lengths (words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()

fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(df_words.loc[df_words['word_count'] <= 2000, 'word_count'],
        bins=40, edgecolor="black")
ax.set(
    title="Distribution of shorter articles (  2 000 words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()

fig, ax = plt.subplots(figsize=(8, 5))
ax.boxplot(df_words['word_count'], vert=False, showfliers=False)
ax.set(
    title="Boxplot of article word counts",
    xlabel="Words per article"
)
plt.show()

df_len = df[['word_count', 'char_count']].copy()

fig, ax = plt.subplots(figsize=(8, 6))
ax.scatter(df_len['word_count'], df_len['char_count'],
           alpha=0.05, s=5)
ax.set(
    title="Characters vs. Words per article",
    xlabel="Words per article",
    ylabel="Characters per article"
)
plt.show()



fig, ax = plt.subplots(figsize=(8, 6))
counts, edges = np.histogram(df_words['word_count'], bins=100)
cdf = np.cumsum(counts) / counts.sum()
ax.plot(edges[1:], cdf)
ax.set(
```

```
    title="CDF - proportion of articles up to length X",
    xlabel="Words per article",
    ylabel="Cumulative proportion"
)
plt.show()


top20 = df.nlargest(20, 'word_count')[['title', 'word_count']].
 ↪reset_index(drop=True)
display(top20)


def simple_tokens(text: str):
    """Lower-cases & grabs alphanum tokens."""
    return re.findall(r'\b\w+\b', text.lower())

sample_size = 50_000                              # down-sample for speed
sample = df.sample(min(sample_size, len(df)), random_state=42)

tok_counter = Counter()
for doc_text in sample['text']:
    tok_counter.update(simple_tokens(doc_text))

top_words = pd.DataFrame(tok_counter.most_common(30),
                         columns=['token', 'freq'])

fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(top_words['token'][::-1], top_words['freq'][::-1])
ax.set(
    title="Top 30 tokens in a 50 k-article sample",
    xlabel="Frequency"
)
plt.show()
```

	title	word_count
0	Chronik der COVID-19-Pandemie in den Vereinigt…	83437
1	Chronik der COVID-19-Pandemie in den Vereinigt…	79832
2	Liste von Filmen mit homosexuellem Inhalt	78666
3	Giacomo Casanova	73974
4	Frankfurt am Main in der Literatur	65648
5	Chronik der COVID-19-Pandemie in den Vereinigt…	63418
6	Bauwerke in Bockenheim	55837
7	Belle Époque	55018
8	Figuren im Star-Trek-Universum	52927
9	Geschichte Osttimors	49132
10	Inhalt und Interpretation der Unendlichen Gesc…	48020
11	Russland	45928
12	Park Klein-Glienicke	41228
13	Geschichte des Saarlandes	39386
14	Scuderia Ferrari	39117
15	Liste der Stolpersteine in Tübingen Innenstadt	38716
16	Die Schlümpfe (Comic-Geschichten)	37116
17	Spätantike	36916
18	Arte 360°-Reportage	36761

19 Figuren im Lied von Eis und Feuer 36098

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#  Build throw-away subsets
df_0_500   = df.loc[df['word_count']       <=  500].copy()
df_500_1k  = df.loc[(df['word_count'] > 500) &
                    (df['word_count'] <= 1000)].copy()
df_0_1k    = df.loc[df['word_count']       <= 1000].copy()


fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharey=True)

# 0-500 words
axes[0].hist(df_0_500['word_count'],
             bins=np.arange(0, 501, 25), edgecolor="black")
axes[0].set(title="Articles 0 - 500 words",
            xlabel="Words per article", ylabel="Number of articles")

# 500-1 000 words
axes[1].hist(df_500_1k['word_count'],
             bins=np.arange(500, 1001, 25), edgecolor="black")
```

```python
axes[1].set(title="Articles 500 - 1 000 words",
            xlabel="Words per article")

plt.tight_layout()
plt.show()


fig, ax = plt.subplots(figsize=(8, 5))
ax.boxplot(
    [df_0_500['word_count'],
     df_500_1k['word_count'],
     df_0_1k['word_count']],
    labels=["0-500", "500-1 000", "0-1 000"],
    vert=False,
    showfliers=False
)
ax.set(title="Word-count distribution by range",
       xlabel="Words per article")
plt.show()


fig, ax = plt.subplots(figsize=(8, 6))
counts, edges = np.histogram(df_0_1k['word_count'], bins=100)
ax.plot(edges[1:], np.cumsum(counts) / counts.sum())
ax.set(title="CDF for articles   1 000 words",
       xlabel="Words per article",
       ylabel="Cumulative proportion")
plt.show()


bands = pd.Series({
    "0-500"      : len(df_0_500),
    "500-1 000"  : len(df_500_1k),
    "1 000-∞"    : len(df) - len(df_0_1k)
})
fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(bands.index, bands.values)
ax.set(title="Article-count by length band",
       ylabel="Number of articles")
for i, v in enumerate(bands.values):
    ax.text(i, v, f"{v:,}", ha="center", va="bottom")
plt.show()
```

```
/var/folders/yx/8v6g52y10p9gxhgxm0dvr2vr0000gn/T/ipykernel_727/1900998789.py:41:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been
renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be
dropped in 3.11.
  ax.boxplot(
```

[ ]:
```python
import re
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np


num_zero = int((df['word_count'] == 0).sum())
print(f"{num_zero} articles have 0 words in the current DataFrame.")


df_short = df[df['word_count'] <= 10].copy()
print(f"{len(df_short)} articles have 0-10 words.")


fig, ax = plt.subplots(figsize=(8, 5))
ax.hist(df_short['word_count'], bins=np.arange(-0.5, 11.5, 1), edgecolor="black")
ax.set(
    title="Distribution of very short articles (0-10 words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()


def simple_tokens(text):
    """Lower-case alphanumeric tokens."""
    return re.findall(r"\b\w+\b", text.lower())

token_counter = Counter()
for text in df_short['text']:
    token_counter.update(simple_tokens(text))

top_tokens = token_counter.most_common(20)
tokens = [t[0] for t in top_tokens][::-1]
freqs = [t[1] for t in top_tokens][::-1]

fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(tokens, freqs)
ax.set(
    title="Top 20 tokens in articles with  10 words",
    xlabel="Frequency"
)
plt.show()
```

```
0 articles have 0 words in the current DataFrame.
237874 articles have 0-10 words.
```

1.4 EDA middel results

The distribution is still majored at the lower end of the word specturm. The outliars at the top still persist. Because I don't know where the semantic cutoff is I don't want to deleate the number of words lower than 10 also because you can see the highest number of articles with under 10 words has 6. wich after some diging arround in the dataset was offen corrolated with family names

1.4.1 Data cleaning middle

in ther EDA you coulöd see until now there where german stopwords in the dataset we can deleate them because they provide no valuble inform,ation to our ebedding I used this topword list: + https://github.com/solariz/german_stopwords

[ ]:
```
from pathlib import Path
import re
from tqdm import tqdm

stopwords_path = Path("PATH")

with open(stopwords_path, 'r', encoding='utf-8') as f:
    custom_german_stopwords = set(line.strip() for line in f if line.strip())

print(f"Loaded {len(custom_german_stopwords)} German stopwords.")

token_re = re.compile(r'\b\w+\b', re.UNICODE)

def clean_text(text: str) -> str:
    tokens = (tok.lower() for tok in token_re.findall(text))
    kept = [tok for tok in tokens if tok not in custom_german_stopwords]
    return " ".join(kept)

tqdm.pandas(desc="Cleaning text with custom stopwords")

df_clean["text"] = df_clean["text"].progress_apply(clean_text)

print(df_clean["text"].head(3))

df = df_clean
```

```
Loaded 594 German stopwords.

Cleaning text with custom stopwords: 100%|      | 2978743/2978743
[05:27<00:00, 9105.63it/s]

0    peter giger 12 april 1939 zürich hans peter gi…
1    uss s 25 ss 130 u boot united states navy holl…
2    gewöhnliche goldrute solidago virgaurea gemein…
Name: text, dtype: object
```

1.5 EDA Nr. 3

After deleating the number of stopwords I updated the number words and chars and performed the EDA one again

[ ]:
```
df['char_count'] = df['text'].apply(len)
df['word_count'] = df['text'].apply(lambda x: len(x.split()))

print(df[['char_count', 'word_count']].describe())

df['word_count'].hist(bins=50)
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import re

# Handy Jupyter settings
%matplotlib inline
plt.style.use("ggplot")

df_words = df[['word_count']].copy()

fig, ax = plt.subplots(figsize=(10, 6))
bins = np.arange(0, 10_001, 500)          # 0-10 k words, 500-word bins
ax.hist(df_words['word_count'], bins=bins, edgecolor="black")
ax.set(
    title="Distribution of article lengths (words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()


fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(df_words.loc[df_words['word_count'] <= 2000, 'word_count'],
        bins=40, edgecolor="black")
ax.set(
    title="Distribution of shorter articles (  2 000 words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()


fig, ax = plt.subplots(figsize=(8, 5))
ax.boxplot(df_words['word_count'], vert=False, showfliers=False)
```

```
ax.set(
    title="Boxplot of article word counts",
    xlabel="Words per article"
)
plt.show()


df_len = df[['word_count', 'char_count']].copy()

fig, ax = plt.subplots(figsize=(8, 6))
ax.scatter(df_len['word_count'], df_len['char_count'],
           alpha=0.05, s=5)               # light dots for big data
ax.set(
    title="Characters vs. Words per article",
    xlabel="Words per article",
    ylabel="Characters per article"
)
plt.show()


fig, ax = plt.subplots(figsize=(8, 6))
counts, edges = np.histogram(df_words['word_count'], bins=100)
cdf = np.cumsum(counts) / counts.sum()
ax.plot(edges[1:], cdf)
ax.set(
    title="CDF - proportion of articles up to length X",
    xlabel="Words per article",
    ylabel="Cumulative proportion"
)
plt.show()

top20 = df.nlargest(20, 'word_count')[['title', 'word_count']].
 ↪reset_index(drop=True)
display(top20)


def simple_tokens(text: str):
    """Lower-cases & grabs alphanum tokens."""
    return re.findall(r'\b\w+\b', text.lower())

sample_size = 50_000                             # down-sample for speed
sample = df.sample(min(sample_size, len(df)), random_state=42)

tok_counter = Counter()
for doc_text in sample['text']:
    tok_counter.update(simple_tokens(doc_text))
```

```
top_words = pd.DataFrame(tok_counter.most_common(30),
                         columns=['token', 'freq'])

fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(top_words['token'][::-1], top_words['freq'][::-1])
ax.set(
    title="Top 30 tokens in a 50 k-article sample",
    xlabel="Frequency"
)
plt.show()
```

	title	word_count
0	Chronik der COVID-19-Pandemie in den Vereinigt…	49144
1	Liste von Filmen mit homosexuellem Inhalt	48691
2	Chronik der COVID-19-Pandemie in den Vereinigt…	47526
3	Chronik der COVID-19-Pandemie in den Vereinigt…	38410
4	Giacomo Casanova	35643
5	Arte 360°-Reportage	35462
6	Bauwerke in Bockenheim	35419
7	Frankfurt am Main in der Literatur	33789
8	Belle Époque	32093
9	Figuren im Star-Trek-Universum	28532
10	Euro-Umlaufmünzen-Motivliste	27767
11	Geschichte Osttimors	27663
12	Russland	26501
13	Liste der von der Hakluyt Society veröffentlic…	23976
14	Die Schlümpfe (Comic-Geschichten)	23079
15	Scuderia Ferrari	22968
16	Geschichte des Saarlandes	22962
17	Inhalt und Interpretation der Unendlichen Gesc…	22567
18	Park Klein-Glienicke	22105

```
19                                      Commodore 128       21463
```

```
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#  Build throw-away subsets
df_0_500   = df.loc[df['word_count']       <=  500].copy()
df_500_1k  = df.loc[(df['word_count'] > 500) &
                    (df['word_count'] <= 1000)].copy()
df_0_1k    = df.loc[df['word_count']       <= 1000].copy()


fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharey=True)

# 0-500 words
axes[0].hist(df_0_500['word_count'],
             bins=np.arange(0, 501, 25), edgecolor="black")
axes[0].set(title="Articles 0 - 500 words",
            xlabel="Words per article", ylabel="Number of articles")

# 500-1 000 words
axes[1].hist(df_500_1k['word_count'],
             bins=np.arange(500, 1001, 25), edgecolor="black")
```

```
axes[1].set(title="Articles 500 - 1 000 words",
            xlabel="Words per article")

plt.tight_layout()
plt.show()


fig, ax = plt.subplots(figsize=(8, 5))
ax.boxplot(
    [df_0_500['word_count'],
     df_500_1k['word_count'],
     df_0_1k['word_count']],
    labels=["0-500", "500-1 000", "0-1 000"],
    vert=False,
    showfliers=False
)
ax.set(title="Word-count distribution by range",
       xlabel="Words per article")
plt.show()


fig, ax = plt.subplots(figsize=(8, 6))
counts, edges = np.histogram(df_0_1k['word_count'], bins=100)
ax.plot(edges[1:], np.cumsum(counts) / counts.sum())
ax.set(title="CDF for articles   1 000 words",
       xlabel="Words per article",
       ylabel="Cumulative proportion")
plt.show()


bands = pd.Series({
    "0-500"      : len(df_0_500),
    "500-1 000"  : len(df_500_1k),
    "1 000-∞"    : len(df) - len(df_0_1k)
})
fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(bands.index, bands.values)
ax.set(title="Article-count by length band",
       ylabel="Number of articles")
for i, v in enumerate(bands.values):
    ax.text(i, v, f"{v:,}", ha="center", va="bottom")
plt.show()
```

```
/var/folders/yx/8v6g52y10p9gxhgxm0dvr2vr0000gn/T/ipykernel_727/1900998789.py:41:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been
renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be
dropped in 3.11.
  ax.boxplot(
```

[ ]:

```
import re
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np


num_zero = int((df['word_count'] == 0).sum())
print(f"{num_zero} articles have 0 words in the current DataFrame.")


df_short = df[df['word_count'] <= 10].copy()
print(f"{len(df_short)} articles have 0-10 words.")


fig, ax = plt.subplots(figsize=(8, 5))
ax.hist(df_short['word_count'], bins=np.arange(-0.5, 11.5, 1),␣
  ↪edgecolor="black")
ax.set(
    title="Distribution of very short articles (0-10 words)",
    xlabel="Words per article",
    ylabel="Number of articles"
)
plt.show()


def simple_tokens(text):
    """Lower-case alphanumeric tokens."""
    return re.findall(r"\b\w+\b", text.lower())

token_counter = Counter()
for text in df_short['text']:
    token_counter.update(simple_tokens(text))

top_tokens = token_counter.most_common(20)
tokens = [t[0] for t in top_tokens][::-1]
freqs = [t[1] for t in top_tokens][::-1]

fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(tokens, freqs)
ax.set(
    title="Top 20 tokens in articles with  10 words",
    xlabel="Frequency"
)
plt.show()
```

```
6 articles have 0 words in the current DataFrame.
307478 articles have 0-10 words.
```

[ ]:
```
word_to_count = "der"
total_der = df["text"].str.count(rf"\b{word_to_count}\b").sum()

print(f'The word "{word_to_count}" appears {total_der:,} times in the corpus.')
```

```
The word "der" appears 3 times in the corpus.
```

[ ]:
```
df.to_parquet("cleaned_wiki_de.parquet", index=False)
```

[ ]:
```
df.head()
```

[ ]:
```
        id                 title  \
0  1873540           Peter Giger
1  1873552         S-25 (U-Boot)
2  1873556  Gewöhnliche Goldrute
3  1873567            Kerckhoffs
4  1873574      Scipione (Maler)

                                                text  char_count  word_count
0  peter giger 12 april 1939 zürich hans peter gi…        3734         512
1  uss s 25 ss 130 u boot united states navy holl…        1559         237
2  gewöhnliche goldrute solidago virgaurea gemein…        3342         350
3   kerckhoffs kerkhoffs familienname personen siehe          48           5
4  scipione gino bonicho 25 februar 1904 macerata…         993         123
```

2 TDIF Vectorisation TEST

Hypothesis: This will not work because TDIF is't rally a semantic algorithm it just looks at the relevance of each words but easyly gets lost one you write truck instead of heavy transport vehicle

I ended up not using it because it wasn't "smart" enough and needent a better sentiment embedding

[ ]:
```
import pandas as pd



df = pd.read_parquet("cleaned_wiki_de.parquet")

print(f"Loaded {len(df):,} articles")
print(df.columns)
print(df.head(2))
```

```
Loaded 2,978,743 articles
Index(['id', 'title', 'text', 'char_count', 'word_count'], dtype='object')
        id        title                                             text  \
0  1873540  Peter Giger  peter giger 12 april 1939 zürich hans peter gi…
```

```
1  1873552  S-25 (U-Boot)  uss s 25 ss 130 u boot united states navy holl…

   char_count  word_count
0        3734         512
1        1559         237
```

[ ]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import vstack
from tqdm import tqdm


batch_size = 1000
docs = df['text'].tolist()


vectorizer = TfidfVectorizer(
    max_features=200_000,
    min_df=5,
    max_df=0.5
)

print("Fitting TF-IDF vocabulary...")
vectorizer.fit(docs)

print("Transforming documents in batches with progress...")
X_batches = []
for i in tqdm(range(0, len(docs), batch_size)):
    batch_docs = docs[i:i+batch_size]
    X_batch = vectorizer.transform(batch_docs)
    X_batches.append(X_batch)

X_tfidf = vstack(X_batches)

print("TF-IDF shape:", X_tfidf.shape)
```

```
Fitting TF-IDF vocabulary…
Transforming documents in batches with progress…

100%|      | 2979/2979 [05:20<00:00,  9.31it/s]

TF-IDF shape: (2978743, 200000)
```

Fitting TF-IDF vocabulary… took 5 Minutes

[ ]:
```python
from scipy.sparse import save_npz

save_npz("tfidf_matrix.npz", X_tfidf)
```

[ ]:
```python
import joblib

joblib.dump(vectorizer, "tfidf_vectorizer.pkl")
```

[ ]: ['tfidf_vectorizer.pkl']

[ ]:
```python
from scipy.sparse import load_npz

X_tfidf = load_npz("tfidf_matrix.npz")
```

2.0.1 Fitting TDIF to the clustering model

[ ]:
```python
import numpy as np
from scipy.sparse import load_npz
import joblib
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from tqdm import tqdm
import matplotlib.pyplot as plt

X_tfidf = load_npz("tfidf_matrix.npz")                  # your (3M × 8.3M) sparse␣
↪matrix
vectorizer = joblib.load("tfidf_vectorizer.pkl")      # fitted TfidfVectorizer
```

[ ]:
```python
svd = TruncatedSVD(n_components=100, random_state=42)
X_reduced = svd.fit_transform(X_tfidf)  # → (3M × 100)
```

[ ]:
```python
def find_best_k(X, k_min=2, k_max=20, sample_size=10000):
    idx = np.random.choice(X.shape[0], size=sample_size, replace=False)
    X_samp = X[idx]

    best_k, best_score = k_min, -1
    scores = {}
    for k in tqdm(range(k_min, k_max+1), desc="Silhouette scan"):
        km = KMeans(n_clusters=k, random_state=42, n_init='auto')
        labels = km.fit_predict(X_samp)
        score = silhouette_score(X_samp, labels)
        scores[k] = score
        if score > best_score:
            best_k, best_score = k, score
    return best_k, best_score, scores

best_k, best_score, scores = find_best_k(X_reduced, k_min=2, k_max=30,␣
↪sample_size=20000)
print(f" Best k = {best_k} (silhouette = {best_score:.4f})")
```

```
Silhouette scan: 100%|      | 29/29 [01:10<00:00,  2.43s/it]
```

```
Best k = 2 (silhouette = 0.6099)
```

```
[ ]: kmeans = KMeans(n_clusters=best_k, random_state=42, n_init='auto')
     labels = kmeans.fit_predict(X_reduced)
```

```
[ ]: terms = vectorizer.get_feature_names_out()
     centroids_orig = kmeans.cluster_centers_ @ svd.components_
     top_terms = []
     for i in range(best_k):
         top_idx = np.argsort(centroids_orig[i])[::-1][:10]
         top_terms.append([terms[j] for j in top_idx])

     pca2 = PCA(n_components=2, random_state=42)
     X_2d = pca2.fit_transform(X_reduced)

     plot_size = 15000
     idx_plot = np.random.choice(X_2d.shape[0], size=plot_size, replace=False)
     fig, ax = plt.subplots(figsize=(12, 10))
     scatter = ax.scatter(
         X_2d[idx_plot,0],
         X_2d[idx_plot,1],
         c=labels[idx_plot],
         alpha=0.5,
         s=5
     )

     centers_2d = pca2.transform(kmeans.cluster_centers_)
     for i, (x,y) in enumerate(centers_2d):
         ax.text(x, y, f"#{i}\n" + ", ".join(top_terms[i][:5]),
                 fontsize=9, weight="bold",
                 bbox=dict(boxstyle="round,pad=0.3", alpha=0.3))

     ax.set_title(f"KMeans clustering (k={best_k}) of Wikipedia articles")
     ax.set_xlabel("PC 1")
     ax.set_ylabel("PC 2")
     plt.tight_layout()
     plt.show()
```

###TDIF results The results with TDIF where underwhelming you yould see a major clustering at the 0,0 point and then only linerar ofshhot groups

3 Now with e5 embedding

the e5 large model is a embedding model wich scores high on the huggingface embedding leaderboard. Espeachily in clustering. The good thing is because it is only a 512b parameter model it works on a moderalty speced hardware

Because the model input is limited I neede to perform further data cleaning and EDA

EDA for e5

[ ]:
```
count_above_2200 = len(df[df['word_count'] > 2200])
print(f"Number of articles with word count > 2200: {count_above_2200}")
```

```
Number of articles with word count > 2200: 15500
```

[ ]:
```
median_above_2200 = df[df['word_count'] > 2200]['word_count'].median()
print(f"Median word count of articles with > 2200 words: {median_above_2200}")
```

```
Median word count of articles with > 2200 words: 3090.0
```

```
[ ]: mean_above_2200 = df[df['word_count'] > 2200]['word_count'].mean()
     print(f"Mean word count of articles with > 2200 words: {mean_above_2200:.2f}")
```

```
Mean word count of articles with > 2200 words: 3817.45
```

Datacleaning for e5 Because the number of tokens is limited I wanted to remove the number wich I think don't hold further value. Namely the numbers 0-99. They where found in the remaining number of words suprisingly often, and shoudn't have much of a meaning because tehy are used in lsits etc.

```
[ ]: from collections import Counter
     import re

     # Combine all text into one big string
     all_text = " ".join(df['text'].astype(str))

     # Tokenize the text: lowercase, remove punctuation, split by whitespace
     words = re.findall(r'\b\w+\b', all_text.lower())

     # Count word frequencies
     word_counts = Counter(words)

     # Get the 100 most common words
     common_words = word_counts.most_common(100)

     # Display as a DataFrame (optional, for readability)
     common_df = pd.DataFrame(common_words, columns=['word', 'count'])
     print(common_df)
```

```
[ ]: import re

     removed_word_count = 0

     def clean_text_and_count(text):
         global removed_word_count
         text = text.lower()
         to_remove = re.findall(r'\b([1-9]?[0-9])\b', text)
         removed_word_count += len(to_remove)
         text = re.sub(r'\b([1-9]?[0-9])\b', '', text)
         text = re.sub(r'\s+', ' ', text).strip()
         return text

     df['text_cleaned'] = df['text'].astype(str).apply(clean_text_and_count)

     print(f" Number of numeric words (0-99) removed: {removed_word_count:,}")
```

```
 Number of numeric words (0-99) removed: 19,897,504
```

Results after datacleaning Another 20 Million chars removed BUt this is still not enough so I loaded the bigger stopwords list and cleaned the Text with it

[ ]:
```
df['word_count_cleaned'] = df['text_cleaned'].apply(lambda x: len(x.split()))

total_words = df['word_count_cleaned'].sum()
print(f" Total words after cleaning: {total_words:,}")
print(df[['word_count', 'word_count_cleaned']].describe())
```

```
 Total words after cleaning: 584,375,019
         word_count  word_count_cleaned
count  2.978743e+06        2.978743e+06
mean   2.028616e+02        1.961818e+02
std    3.863599e+02        3.741002e+02
min    0.000000e+00        0.000000e+00
25%    4.900000e+01        4.700000e+01
50%    1.170000e+02        1.130000e+02
75%    2.290000e+02        2.210000e+02
max    4.914400e+04        4.830700e+04
```

```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the
↪failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```

[ ]:
```
count_above_2200 = len(df[df['word_count_cleaned'] > 2200])
print(f"Number of articles with word count > 2200: {count_above_2200}")
```

```
Number of articles with word count > 2200: 14667
```

[ ]:
```
median_above_2200 = df[df['word_count_cleaned'] > 2200]['word_count_cleaned'].
↪median()
print(f"Median word count of articles with > 2200 words: {median_above_2200}")
```

```
Median word count of articles with > 2200 words: 3072.0
```

[ ]:
```
mean_above_2200 = df[df['word_count_cleaned'] > 2200]['word_count_cleaned'].
↪mean()
print(f"Mean word count of articles with > 2200 words: {mean_above_2200:.2f}")
```

```
Mean word count of articles with > 2200 words: 3786.59
```

[ ]:
```python
from collections import Counter
import re
from itertools import chain

def tokenize(text):
    return re.findall(r'\b\w+\b', text)

token_lists = df['text_cleaned'].apply(tokenize)
all_words = list(chain.from_iterable(token_lists))

word_counts = Counter(all_words)
common_words = word_counts.most_common(20)

common_df = pd.DataFrame(common_words, columns=['word', 'count'])
print(common_df)
```

```
          word    count
0         jahr  1797863
1         zwei  1339663
2        jahre  1235306
3       jahren  1096519
4       ersten  1080875
5           de   939286
6         zeit   910096
7         drei   909104
8         teil   875735
9        stadt   870282
10       leben   866833
11          of   861263
12      später   857423
13  geschichte   801048
14        ende   724448
15       erste   720985
16       liegt   713258
17   deutschen   711784
18       heute   687365
19           a   684384
```

[ ]:
```python
from collections import Counter
import re

all_text = " ".join(df['text'].astype(str))

words = re.findall(r'\b\w+\b', all_text.lower())

word_counts = Counter(words)

common_words = word_counts.most_common(100)
```

```
common_df = pd.DataFrame(common_words, columns=['word', 'count'])
print(common_df)
```

```
         word    count
0        jahr  1797863
1           1  1697421
2        zwei  1339663
3       jahre  1235306
4           2  1170276
..        …        …
95      meter   389095
96       ohne   386480
97   erstmals   380210
98  deutscher   374343
99         st   372400

[100 rows x 2 columns]
```

[ ]:
```
from pathlib import Path
import re
from tqdm import tqdm

stopwords_path = Path("german_stopwords_full.txt")

with open(stopwords_path, 'r', encoding='utf-8') as f:
    custom_german_stopwords = set(line.strip() for line in f if line.strip())

print(f"Loaded {len(custom_german_stopwords)} German stopwords.")

token_re = re.compile(r'\b\w+\b', re.UNICODE)

def clean_text(text: str) -> str:
    tokens = (tok.lower() for tok in token_re.findall(text))
    kept = [tok for tok in tokens if tok not in custom_german_stopwords]
    return " ".join(kept)

tqdm.pandas(desc="Cleaning text with custom stopwords")

df["text_cleaned"] = df["text_cleaned"].progress_apply(clean_text)

print(df["text_cleaned"].head(3))
```

```
Loaded 1861 German stopwords.

Cleaning text with custom stopwords: 100%|      | 2978743/2978743
[02:46<00:00, 17922.23it/s]

0    peter giger april 1939 zürich hans peter giger…
```

```
1    uss s ss 130 u boot united states navy holland…
2    gewöhnliche goldrute solidago virgaurea gemein…
Name: text_cleaned, dtype: object
```

Saving the output to not have to do the long process again and again

```
[ ]: print(df.head())
```

```
        id                  title  \
0  1873540            Peter Giger
1  1873552           S-25 (U-Boot)
2  1873556  Gewöhnliche Goldrute
3  1873567             Kerckhoffs
4  1873574       Scipione (Maler)

                                                text  char_count  word_count  \
0  peter giger 12 april 1939 zürich hans peter gi…        3734         512
1  uss s 25 ss 130 u boot united states navy holl…        1559         237
2  gewöhnliche goldrute solidago virgaurea gemein…        3342         350
3   kerckhoffs kerkhoffs familienname personen siehe          48           5
4  scipione gino bonicho 25 februar 1904 macerata…         993         123

                                        text_cleaned
0  peter giger april 1939 zürich hans peter giger…
1  uss s ss 130 u boot united states navy holland…
2  gewöhnliche goldrute solidago virgaurea gemein…
3         kerckhoffs kerkhoffs familienname personen
4  scipione gino bonicho februar 1904 macerata no…
```

```
[ ]: df = df.drop(columns=["text", "word_count", "char_count"])
```

```
[ ]: df.to_parquet("wikipedia_stream.parquet", index=False)
```

3.0.1 Dataclening to 500 tokens+

because the model can only take 514 tokens I needed to prune each text to fit into the token limit (For this I had Chatgpt help me)

Because I had to tokenise the entire text to do that (I think) I first limited the number of words/characters the Text could have and cut of after the first 450. The intorduction with the most valuble information is tipicly saved tehre so I should still have semantics intact.

EDA for Tokens At first I thought jsut limiting the number of Chars to 350 would solve the issue the problem with that is the long German Words . you will see some graphs below wich display the number of tokens to the number of words.

So the only feasible way to combat this was to cutt of at the embedding level

```
[ ]: import pyarrow.parquet as pq
```

```python
pq_file = pq.ParquetFile("wikipedia_stream.parquet")
print("Num Row Groups:", pq_file.num_row_groups)
print("Num Rows:", pq_file.metadata.num_rows)
print("Columns:", pq_file.schema)
```

```
Num Row Groups: 3
Num Rows: 2978743
Columns: <pyarrow._parquet.ParquetSchema object at 0x371f1b4c0>
required group field_id=-1 schema {
  optional binary field_id=-1 id (String);
  optional binary field_id=-1 title (String);
  optional binary field_id=-1 text_cleaned (String);
}
```

[ ]:
```python
import pandas as pd

df = pd.read_parquet("wikipedia_stream.parquet")
total_limited_char_count = df["text_cleaned"].str.len().clip(upper=1000).sum()
print(total_limited_char_count)
```

[ ]:
```python
import pandas as pd

df = pd.read_parquet("wikipedia_stream.parquet")
def cap_at_350_words(text):
    return ' '.join(text.split()[:450])

df['text_350_words'] = df['text_cleaned'].apply(cap_at_350_words)
```

[ ]:
```python
import matplotlib.pyplot as plt

over_limit_count = (df['token_count_350'] > 514).sum()

print(f"Number of articles over 514 tokens: {over_limit_count}")

plt.figure(figsize=(10, 6))
plt.hist(df['token_count_350'], bins=50, edgecolor='black')
plt.axvline(x=514, color='red', linestyle='--', label='Token limit (514)')
plt.title('Distribution of Token Counts (350-word intros)')
plt.xlabel('Token count')
plt.ylabel('Number of articles')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Number of articles over 514 tokens: 637311
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

df['char_count_350'] = df['text_350_words'].str.len()

token_char_stats = df.groupby('token_count_350')['char_count_350'].agg(['mean',␣
↪'count']).reset_index()

plt.figure(figsize=(10, 6))
plt.plot(token_char_stats['token_count_350'], token_char_stats['mean'],␣
↪marker='o')
plt.title('Average Character Count vs. Token Count (350-word samples)')
plt.xlabel('Token Count')
plt.ylabel('Average Character Count')
plt.grid(True)
plt.tight_layout()
plt.show()
```

[ ]:
```
range_df = df[(df['token_count_350'] >= 490) & (df['token_count_350'] <= 514)]

average_chars = range_df['char_count_350'].mean()

print(f" Average character count for token length 490-514: {average_chars:.
↪2f}")
```

```
 Average character count for token length 490-514: 2015.70
```

[ ]:
```
def cap_at_2015_chars_preserve_words(text):
    if len(text) <= 2015:
        return text
    trimmed = text[:2015]
    last_space = trimmed.rfind(' ')
    return trimmed[:last_space] if last_space != -1 else trimmed

df['text_2015char_clean'] = df['text_350_words'].
↪apply(cap_at_2015_chars_preserve_words)
```

[ ]:
```
from transformers import AutoTokenizer
from tqdm import tqdm
import numpy as np

tokenizer = AutoTokenizer.from_pretrained("intfloat/
↪multilingual-e5-large-instruct")
```

```
texts = df['text_2015char_clean'].tolist()

batch_size = 512
token_counts = []

for i in tqdm(range(0, len(texts), batch_size), desc="Batch tokenizing"):
    batch = texts[i:i+batch_size]
    encoded = tokenizer(batch, truncation=False, padding=False,␣
↪add_special_tokens=True)
    lengths = [len(ids) for ids in encoded['input_ids']]
    token_counts.extend(lengths)

df['token_count_350_char'] = token_counts
```

```
Batch tokenizing:   0%|          | 0/5818 [00:00<?, ?it/s]Token indices sequence
length is longer than the specified maximum sequence length for this model (529
> 512). Running this sequence through the model will result in indexing errors
Batch tokenizing: 100%|     | 5818/5818 [06:12<00:00, 15.63it/s]
```

[ ]:
```
import matplotlib.pyplot as plt

over_limit_count = (df['token_count_350_char'] > 514).sum()

print(f"Number of articles over 514 tokens: {over_limit_count}")

plt.figure(figsize=(10, 6))
plt.hist(df['token_count_350_char'], bins=50, edgecolor='black')
plt.axvline(x=514, color='red', linestyle='--', label='Token limit (514)')
plt.title('Distribution of Token Counts (350-word intros)')
plt.xlabel('Token count')
plt.ylabel('Number of articles')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Number of articles over 514 tokens: 237869
```

[ ]:
```python
from transformers import AutoTokenizer
from tqdm import tqdm

tokenizer = AutoTokenizer.from_pretrained("intfloat/
    ↪multilingual-e5-large-instruct")

def truncate_batch(texts, max_tokens=500):
    encodings = tokenizer(
        texts,
        return_offsets_mapping=True,
        truncation=False,
        padding=False,
        add_special_tokens=True
    )

    truncated_texts = []
    for text, offsets, input_ids in zip(texts, encodings['offset_mapping'],␣
    ↪encodings['input_ids']):
        if len(input_ids) <= max_tokens:
            truncated_texts.append(text)
        else:
            last_char_pos = offsets[max_tokens - 1][1]
            truncated_texts.append(text[:last_char_pos].rstrip())
    return truncated_texts
```

```
BATCH_SIZE = 1024

def batched_truncate(df, column="text_350_words"):
    results = []
    for i in tqdm(range(0, len(df), BATCH_SIZE), desc="Batch Tokenizing"):
        batch = df[column].iloc[i:i + BATCH_SIZE].tolist()
        truncated = truncate_batch(batch)
        results.extend(truncated)
    return results

df['text_exact_500'] = batched_truncate(df)

print(df.head)
```

```
c:\Users\Xperion\AppData\Local\Programs\Python\Python313\Lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
None of PyTorch, TensorFlow >= 2.0, or Flax have been found. Models won't be
available and only tokenizers, configuration and file/data utilities can be
used.
Batch Tokenizing:   0%|          | 0/5818 [00:00<?, ?it/s]Token indices sequence
length is longer than the specified maximum sequence length for this model (869
> 512). Running this sequence through the model will result in indexing errors
Batch Tokenizing: 100%|      | 5818/5818 [07:06<00:00, 13.63it/s]
```

[ ]:
```
import pandas as pd

df.to_parquet("wikidump_514.parquet", index=False)
```

[ ]:
```
from transformers import AutoTokenizer
from tqdm import tqdm
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_parquet("wikidump_514.parquet", columns=['text_exact_514'])

tokenizer = AutoTokenizer.from_pretrained("intfloat/
 ↪multilingual-e5-large-instruct")

texts = df['text_exact_514'].tolist()
batch_size = 512
token_counts = []

for i in tqdm(range(0, len(texts), batch_size), desc="Counting tokens in␣
 ↪batches"):
```

```
    batch = texts[i:i+batch_size]
    encoded = tokenizer(batch, truncation=False, padding=False,␣
↪add_special_tokens=True)
    lengths = [len(ids) for ids in encoded['input_ids']]
    token_counts.extend(lengths)

df['token_count_exact_514'] = token_counts

plt.figure(figsize=(10, 6))
plt.hist(df['token_count_exact_514'], bins=50, edgecolor='black')
plt.title('Token Count Distribution for text_exact_514')
plt.xlabel('Token Count')
plt.ylabel('Number of Articles')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Counting tokens in batches:   0%|          | 0/5818 [00:00<?, ?it/s]Token
indices sequence length is longer than the specified maximum sequence length for
this model (515 > 512). Running this sequence through the model will result in
indexing errors
Counting tokens in batches: 100%|     | 5818/5818 [05:58<00:00, 16.23it/s]
```

3.0.2 Resusts Now I have a Workable Text set now onto the embedding

Because of the size of the dataset every step took some time but the time for the embedding had me floored. It should take 200h so I decided I would only process the first half of my dataset from here on foreward. With a more Powerful GPU this would have been possible with the entire dataset, unfortionatly I am not lucky enough to have a A100

[ ]:
```
df = pd.read_parquet("wikidump_514.parquet")
print(df.head)
```

[ ]:
```
print(df.columns)
```

```
Index(['id', 'title', 'text_cleaned', 'text_350_words', 'text_exact_500'],
dtype='object')
```

[ ]:
```
import os
import pandas as pd
from sentence_transformers import SentenceTransformer
from tqdm import tqdm
import torch

df = pd.read_parquet("wikidump_514.parquet")

df = df.iloc[:len(df) // 2]

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f" Using device: {device}")

model = SentenceTransformer("intfloat/multilingual-e5-large-instruct",␣
↪device=device)

texts = ["passage: " + t for t in df['text_exact_500'].tolist()]

batch_size = 64
embeddings = []

for i in tqdm(range(0, len(texts), batch_size), desc="Embedding batch"):
    batch = texts[i:i + batch_size]
    batch_embeddings = model.encode(
        batch,
        batch_size=batch_size,
        show_progress_bar=False,
        device=device
    )
    embeddings.extend(batch_embeddings)
```

```
df["embedding"] = embeddings

os.makedirs("chunks_50_embedding", exist_ok=True)
df.to_parquet("chunks_50_embedding/wikidump_part_01_with_embedding.parquet",␣
 ↪index=False)

print("  Embedding complete and saved.")
```

3.0.3 Time

This Embedding for half of df took on a 4090 7 hours

4 Unsupervised Clustering

4.1 Kmeans

[2]:
```
import pandas as pd
df = pd.read_parquet("wikidump_part_01_with_embedding.parquet")

print(df.columns)
print(df.head(2))
```

```
Index(['id', 'title', 'text_cleaned', 'text_350_words', 'text_exact_500',
       'embedding'],
      dtype='object')
        id          title                                       text_cleaned  \
0  1873540    Peter Giger  peter giger april 1939 zürich hans peter giger…
1  1873552  S-25 (U-Boot)  uss s ss 130 u boot united states navy holland…

                                     text_350_words  \
0  peter giger april 1939 zürich hans peter giger…
1  uss s ss 130 u boot united states navy holland…

                                     text_exact_500  \
0  peter giger april 1939 zürich hans peter giger…
1  uss s ss 130 u boot united states navy holland…

                                          embedding
0  [0.017257495, 0.0020586113, -0.035239954, -0.0…
1  [0.01616242, -0.0037445973, -0.033856593, -0.0…
```

[3]:
```
df = df.drop(columns=["text_cleaned", "id", "text_350_words", "text_exact_500"])
```

[4]:
```
df.to_parquet("wikidump_half_title_embedding.parquet", index=False)
```

[2]:
```
import pandas as pd
df = pd.read_parquet("wikidump_half_title_embedding.parquet")
```

```
print(df.columns)
print(df.head(2))
```

```
Index(['title', 'embedding'], dtype='object')
           title                                          embedding
0    Peter Giger  [0.017257495, 0.0020586113, -0.035239954, -0.0…
1  S-25 (U-Boot)  [0.01616242, -0.0037445973, -0.033856593, -0.0…
```

[ ]:
```
import faiss
import numpy as np
import pandas as pd
from sklearn.cluster import DBSCAN

embeddings = np.vstack(df['embedding'].values).astype('float32')
```

[ ]:
```
faiss.normalize_L2(embeddings)
```

[5]:
```
import numpy as np
import pandas as pd
import faiss
from tqdm import tqdm
# Set number of clusters
n_clusters = 15000
kmeans = faiss.Kmeans(d=embeddings.shape[1], k=n_clusters, niter=20,␣
 ↪verbose=True)
kmeans.train(embeddings)

# Search nearest cluster center for each vector (with tqdm progress)
batch_size = 100_000
cluster_ids = []

index = kmeans.index
for i in tqdm(range(0, embeddings.shape[0], batch_size), desc="Assigning␣
 ↪clusters"):
    end = min(i + batch_size, embeddings.shape[0])
    _, I = index.search(embeddings[i:end], 1)
    cluster_ids.extend(I.flatten())

df['cluster'] = cluster_ids
```

```
Clustering 1489371 points in 1024D to 15000 clusters, redo 1 times, 20
iterations
  Preprocessing in 0.81 s
  Iteration 19 (674.49 s, search 671.41 s): objective=145412 imbalance=1.293
nsplit=0
```

```
Assigning clusters: 100%|      | 15/15 [00:33<00:00,  2.27s/it]
```

[ ]:
```python
top_articles = (
    df.groupby('cluster')['title']
    .apply(lambda x: x.sample(n=min(5, len(x)), random_state=42).tolist())
    .reset_index(name='example_titles')
)

print(top_articles.head(50))
```

```
    cluster                                      example_titles
0         0  [Blagoje Marjanović, Siniša Oreščanin, Nikola …
1         1  [Louis Frédéric Berger, Jean-Philippe Dardier,…
2         2       [Pröbster, Preglau, Pronk, Prückner, Printz]
3         3  [Michael Herr (Biathlet), Deutsche Meisterscha…
4         4        [Rejštejn, Rochov, Rosička, Rozkoš, Rožnov]
5         5  [Jonas Panamariovas, Ramūnas Karbauskis, Lietu…
6         6  [Jogaku Zasshi, Kodomo, Robot (Manga), Chūgoku…
7         7  [Saucedilla, Campillo de Altobuey, Burujón, Na…
8         8  [Abū Righāl, Ichwān, Musᶜab ibn ᶜUmair, Hunain…
9         9  [Výrovice, Zdechovice u Nového Bydžova, Nyklov…
10       10  [NOON-Zustand, Mikrokanonisches Ensemble, Dyso…
11       11  [Liste der Studentenverbindungen in Paderborn,…
12       12  [Gramm-Bernstein Motor Truck Corporation, Jewe…
13       13  [Hydrocephalus, Pinealiszyste, Astroblastom, P…
14       14  [Florida, Dry Tortugas, Islamorada, Anhinga Tr…
15       15  [Immanuel Ngatjizeko, Eduard Afrikaner, John Y…
16       16  [Alleyway, Super Mario Galaxy, Bounce (Compute…
17       17  [North Shropshire, Tandridge District, Mendip,…
18       18                         [TDI, Bukh, MAQ, KOQ, AZZ]
19       19  [Hans Morgenthaler, Gustav von Reymond, Johann…
20       20  [Zach Bogosian, Chris Wideman, Kyle Okposo, Ty…
21       21  [James Beattie (Schriftsteller), Nicolas de la…
22       22  [1. Divisjon 1986, Tippeligaen 2005, Hovedseri…
23       23  [Ordulf (Sachsen), Otto II. (Braunschweig-Lüne…
24       24  [Audun, Français, Ussel, Malzieu, Montauban (B…
25       25  [O'Flaherty-Clan, Charles O'Brien de Clare, D'…
26       26             [Noguer, Nivert, Niblo, Nejar, Nocke]
27       27  [Dunkelziffer der Armut, Umlage U2, Kindergeld…
28       28  [Hallein (Stadtteil), Salzsiedepfanne, Thoßfel…
29       29  [Credit Default Swap Index, Single-Index-Model…
30       30  [Metzlesberg, Sommerau (Feuchtwangen), Hohensc…
31       31  [Tabakmottenschildlaus, Kleiderlaus, Kopflaus,…
32       32  [Jomtov Benjaes, Judah Touro, Samuel Levi (Cha…
33       33  [Ballyhaunis, Leighlinbridge, County Kilkenny,…
34       34  [Creutzwald, Boulay-Moselle, Téterchen, Apach,…
35       35  [Marquess of Ripon, Earl of Yarborough, Earl B…
36       36  [OKATO, Altbolschewik, Sowdepien, Autonomer Kr…
37       37  [Qark Fier, Golloborda, Libohova, Pazari i Ri,…
38       38  [Remigiusberg-Formation, Kupferschiefer, Obere…
```

```
39       39  [Susanne-Stellung, Narew-Offensive, Unternehme…
40       40  [It's the End of the World as We Know It (And …
41       41        [Bös, Heuschkel, Piroch, Haefliger, Zirner]
42       42  [Beniamino Cesi, Girolamo Crescentini, Giusepp…
43       43  [Mikrokügelchen, Blotting, Sequenzierautomat, …
44       44  [Souhayr Belhassen, Amir Ashour, Najla Kassab,…
45       45  [Maigesetze (Österreich-Ungarn), Kurhessische …
46       46  [Liste der Kulturgüter in Dübendorf, Liste der…
47       47  [Paige Culver, Ashley Rodrigues, Kanadische Fu…
48       48  [Wilhelm Schneider (Politiker, 1915), Otto Bey…
49       49  [Bruce Pirnie, Jim Driscoll (Leichtathlet), Da…
```

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from tqdm import tqdm

embeddings = np.vstack(df['embedding'].values).astype('float32')

import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
import plotly.express as px



# Reduce to 2D using PCA
pca = PCA(n_components=2)
embedding_2d = pca.fit_transform(embeddings)

# Add to DataFrame
df["x"] = embedding_2d[:, 0]
df["y"] = embedding_2d[:, 1]
```

[10]:
```python
print(df.head(2))
```

```
          title                                          embedding  cluster  \
0    Peter Giger  [0.017257495, 0.0020586113, -0.035239954, -0.0…     8266
1  S-25 (U-Boot)  [0.01616242, -0.0037445973, -0.033856593, -0.0…     9302

         x         y
0 -0.094429 -0.092861
1 -0.085997  0.019098
```

[11]:
```python
df.to_parquet("wikidump_half_title_embedding_cluster_x_y.parquet", index=False)
```

[2]:
```python
import pandas as pd
df = pd.read_parquet("wikidump_half_title_embedding_cluster_x_y.parquet")
```

[ ]:
```python
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species")
fig.show()
```

[3]:
```python
import plotly.express as px

fig = px.scatter(
    df.sample(100_000),
    x="x", y="y",
    color="cluster",
    hover_data=["title"],
    title="Wikipedia Clusters in 2D",
    width=1000, height=800
)
fig.show()
```

[ ]:
```python
import plotly.express as px
import numpy as np

random_clusters = np.random.choice(df['cluster'].unique(), size=2,␣
 ↪replace=False)

filtered_df = df[df['cluster'].isin(random_clusters)]

filtered_df = filtered_df.sample(min(10000, len(filtered_df)))

fig = px.scatter(
    filtered_df,
    x="x", y="y",
    color="cluster",
    hover_data=["title"],
    title=f"Comparison of 2 Random Wikipedia Clusters: {random_clusters[0]} vs␣
 ↪{random_clusters[1]}",
    width=1000, height=800
)
fig.show()
```

[7]:
```python
df = pd.read_parquet("wikidump_half_title_embedding_cluster_x_y.parquet")
```

[ ]:
```python
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
import plotly.express as px
```

```
selected_clusters = np.random.choice(df['cluster'].unique(), size=10,␣
↪replace=False)

samples_per_cluster = 1000
filtered_df = pd.concat([
    df[df['cluster'] == cluster].sample(n=min(samples_per_cluster,␣
↪len(df[df['cluster'] == cluster])), random_state=42)
    for cluster in selected_clusters
])

pca = PCA(n_components=3)
embedding_3d = pca.fit_transform(np.vstack(filtered_df['embedding'].values).
↪astype('float32'))

filtered_df["x3d"] = embedding_3d[:, 0]
filtered_df["y3d"] = embedding_3d[:, 1]
filtered_df["z3d"] = embedding_3d[:, 2]

fig = px.scatter_3d(
    filtered_df,
    x="x3d", y="y3d", z="z3d",
    color="cluster",
    hover_data=["title"],
    title="3D Visualization of 10 Random Wikipedia Clusters",
    width=1000, height=800
)
fig.show()
```

4.1.1 Results of K-Means

AS you could see the K-means Clustering provided workable clusters. With a bit of manual Validation many of the clusters are correct or mostly correct. The biggest Problem of the M-Means algorithm is that you have to define the number of clusters. I chose 15.000 Clusters after some experementing because that provided good results with less wrong articles in a cluster. But I also did't want to have way to many clusters.

Mabe there is something better? Because of havong to set the number of Clusters the performance was not the best clustering possible. Next I wanted to use HDBSCAN to find out the number of clusters Dynamicly

4.2 HDBSCAN

Because performing hdbscan on my hardware with 1000 dimensions was not possible I will bring down the number of dimensions down to 30, first with fast PCA and than more granualr with UMAP. After that I displayed a few exambles of the clusters in graphs. More on that in the results section.

[ ]:
```python
import pandas as pd
import numpy as np
import umap
import hdbscan
import matplotlib.pyplot as plt
from tqdm import tqdm
import time

print("  Loading data...")
df = pd.read_parquet("wikidump_half_title_embedding.parquet")

print("  Converting embeddings to NumPy array...")
start = time.time()
embeddings = np.vstack(df['embedding'].values).astype('float32')
print(f"  Done in {time.time() - start:.2f} seconds.")
```

```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
 Loading data…
 Converting embeddings to NumPy array…
 Done in 35.82 seconds.
```

[ ]:
```python
import time
from sklearn.decomposition import PCA
from umap import UMAP
import hdbscan

# Reduce input dims from 1000 to 100
print("  Running PCA...")
pca = PCA(n_components=100, random_state=42)
embeddings_pca = pca.fit_transform(embeddings)

# Step 2: UMAP 10D
print("  Running UMAP...")
umap_model = UMAP(
    n_components=10,
    n_neighbors=30,
    metric='cosine',
    random_state=42,
    verbose=True,
    n_jobs=-1  # Use all cores
)
X_umap = umap_model.fit_transform(embeddings_pca)
```

```
for i in range(X_umap.shape[1]):
    df[f'umap_{i}'] = X_umap[:, i]

print("  Running HDBSCAN...")
start = time.time()
clusterer = hdbscan.HDBSCAN(min_cluster_size=200, metric='euclidean')
labels = clusterer.fit_predict(X_umap)
print(f"  HDBSCAN done in {time.time() - start:.2f} seconds.")

df['hdbscan_cluster'] = labels

output_path = "wikidump_half_title_embedding_cluster_umap.parquet"
df.to_parquet(output_path, index=False)
print(f"  Saved clustered DataFrame with UMAP to: {output_path}")
```

```
 Running PCA…
 Running UMAP…
UMAP(angular_rp_forest=True, metric='cosine', n_components=10, n_jobs=1,
n_neighbors=30, random_state=42, verbose=True)
```

```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was
renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```

```
Fri Apr 25 10:24:58 2025 Construct fuzzy simplicial set
Fri Apr 25 10:24:59 2025 Finding Nearest Neighbors
Fri Apr 25 10:24:59 2025 Building RP forest with 64 trees
Fri Apr 25 10:26:23 2025 NN descent for 21 iterations
         1  /  21
         2  /  21
         3  /  21
         4  /  21
        Stopping threshold met -- exiting after 4 iterations
Fri Apr 25 10:31:03 2025 Finished Nearest Neighbor Search
Fri Apr 25 10:31:12 2025 Construct embedding
```

[ ]:
```
import time
import umap.umap_ as umap
from sklearn.decomposition import PCA
from umap import UMAP

# Reduce input dims from 1000 to 100
pca = PCA(n_components=100, random_state=42)
```

```python
embeddings_pca = pca.fit_transform(embeddings)

umap_model = UMAP(n_components=10, n_neighbors=30, metric='cosine',␣
 ↪random_state=42, verbose=True)
X_umap = umap_model.fit_transform(embeddings_pca)
```

```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was
renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```

```
UMAP(angular_rp_forest=True, metric='cosine', n_components=10, n_jobs=1,
n_neighbors=30, random_state=42, verbose=True)
Fri Apr 25 00:27:25 2025 Construct fuzzy simplicial set
Fri Apr 25 00:27:25 2025 Finding Nearest Neighbors
Fri Apr 25 00:27:25 2025 Building RP forest with 64 trees
Fri Apr 25 00:28:52 2025 NN descent for 21 iterations
         1  /  21
         2  /  21
         3  /  21
         4  /  21
        Stopping threshold met -- exiting after 4 iterations
Fri Apr 25 00:33:53 2025 Finished Nearest Neighbor Search
Fri Apr 25 00:34:03 2025 Construct embedding
```

```
Epochs completed:   0%|            1/200 [00:00]

        completed  0  /  200 epochs

Epochs completed:  10%|            21/200 [02:26]

        completed  20  /  200 epochs

Epochs completed:  20%|            41/200 [05:20]

        completed  40  /  200 epochs

Epochs completed:  30%|            61/200 [08:15]

        completed  60  /  200 epochs

Epochs completed:  40%|            81/200 [11:12]

        completed  80  /  200 epochs

Epochs completed:  50%|           101/200 [14:08]

        completed  100  /  200 epochs

Epochs completed:  60%|           121/200 [17:04]
```

```
        completed  120  /  200 epochs
Epochs completed:  70%|         141/200 [20:01]
        completed  140  /  200 epochs
Epochs completed:  80%|        161/200 [22:57]
        completed  160  /  200 epochs
Epochs completed:  91%|        182/200 [26:03]
        completed  180  /  200 epochs
Epochs completed: 100%|        200/200 [28:42]
Fri Apr 25 09:19:10 2025 Finished embedding
```

[ ]:
```
print("  Running HDBSCAN...")
start = time.time()
clusterer = hdbscan.HDBSCAN(min_cluster_size=200, metric='euclidean')
labels = clusterer.fit_predict(X_umap)
print(f"  HDBSCAN done in {time.time() - start:.2f} seconds.")

# Assign labels
df['hdbscan_cluster'] = labels
```

```
  Running HDBSCAN…
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was
renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-
packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was
renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
Python(17457) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
Python(17458) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
Python(17459) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
Python(17460) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
Python(17461) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
  HDBSCAN done in 264.40 seconds.
```

[ ]:
```
import time
import cupy as cp
import cudf
```

```
from cuml.decomposition import PCA
from cuml.manifold import UMAP

# Load data
print("  Loading data...")
import pandas as pd
import numpy as np
df = pd.read_parquet("wikidump_half_title_embedding.parquet")

# Convert embeddings to CuPy / cuDF
print("  Converting embeddings to GPU format...")
start = time.time()
embeddings_np = np.vstack(df['embedding'].values).astype('float32')
embeddings_gpu = cp.asarray(embeddings_np)  # OR cudf.DataFrame(embeddings_np)
print(f"  Done in {time.time() - start:.2f} seconds.")

# CA on GPU
print("  Running PCA on GPU...")
start = time.time()
pca = PCA(n_components=100, random_state=42)
embeddings_pca_gpu = pca.fit_transform(embeddings_gpu)
print(f"  PCA done in {time.time() - start:.2f} seconds.")

# UMAP on GPU
print("  Running UMAP on GPU...")
start = time.time()
umap_model = UMAP(n_components=10, n_neighbors=30, metric='cosine',␣
 ↪random_state=42, verbose=True)
X_umap_gpu = umap_model.fit_transform(embeddings_pca_gpu)
print(f"  UMAP done in {time.time() - start:.2f} seconds.")
```

```
 Loading data…
 Converting embeddings to GPU format…
 Done in 12.16 seconds.
 Running PCA on GPU…
 PCA done in 0.95 seconds.
 Running UMAP on GPU…
[2025-04-25 13:36:41.413] [CUML] [info] build_algo set to brute_force_knn
because random_state is given
[2025-04-25 13:36:41.422] [CUML] [debug] Computing KNN Graph
[2025-04-25 13:37:33.346] [CUML] [debug] Computing fuzzy simplicial set
 UMAP done in 65.47 seconds.
```

[2]:
```
df.to_parquet("wikidump_half_title_embedding_cluster_umap.parquet", index=False)
```

[3]:
```
import pandas as pd
```

```python
df = pd.read_parquet("wikidump_half_title_embedding_cluster_umap.parquet")
```

[3]:
```python
from cuml.cluster import HDBSCAN
clusterer = HDBSCAN(min_cluster_size=30)
labels = clusterer.fit_predict(X_umap_gpu)
```

[4]:
```python
labels_np = cp.asnumpy(labels)

df['cluster'] = labels_np
```

[4]:
```python
print(df.head(3))
```

```
                    title                                          embedding  \
0             Peter Giger  [0.017257495, 0.0020586113, -0.035239954, -0.0…
1           S-25 (U-Boot)  [0.01616242, -0.0037445973, -0.033856593, -0.0…
2   Gewöhnliche Goldrute  [0.016481167, 0.018800229, -0.014906349, -0.06…

   cluster
0     1500
1     1701
2     1599
```

[5]:
```python
df.to_parquet("wikidump_clustered.parquet", index=False)
```

[ ]:
```python
import matplotlib.pyplot as plt
import cupy as cp

# Get 2D coordinates
X_plot = cp.asnumpy(X_umap_gpu[:, :2])
labels_cpu = cp.asnumpy(labels)

plt.figure(figsize=(12, 8))
plt.scatter(X_plot[:, 0], X_plot[:, 1], c=labels_cpu, cmap='tab20', s=2)
plt.colorbar(label="Cluster ID")
plt.title("Wikipedia Embedding Clusters (UMAP + Clustering)")
plt.xlabel("UMAP-1")
plt.ylabel("UMAP-2")
plt.show()
```

[ ]:
```
import pandas as pd
import cupy as cp

df['cluster'] = cp.asnumpy(labels)
df['x'] = cp.asnumpy(X_umap_gpu[:, 0])
df['y'] = cp.asnumpy(X_umap_gpu[:, 1])
```

[ ]:
```
import plotly.express as px
import numpy as np

# 2
random_clusters = np.random.choice(df['cluster'].unique(), size=2,␣
↪replace=False)

filtered_df = df[df['cluster'].isin(random_clusters)]

# downsample because otherwise I just wont load/ use way to much ram
filtered_df = filtered_df.sample(min(10000, len(filtered_df)))

fig = px.scatter(
    filtered_df,
```

```
    x="x", y="y",
    color="cluster",
    hover_data=["title"],
    title=f"Comparison of 2 Random Wikipedia Clusters: {random_clusters[0]} vs␣
↪{random_clusters[1]}",
    width=1000, height=800
)
fig.show()
```

[ ]:
```
import plotly.express as px
import numpy as np

# 6
random_clusters = np.random.choice(df['cluster'].unique(), size=6,␣
↪replace=False)
filtered_df = df[df['cluster'].isin(random_clusters)]

# downsample because otherwise I just wont load/ use way to much ram
filtered_df = filtered_df.sample(min(10000, len(filtered_df)))

# Plot
fig = px.scatter(
    filtered_df,
    x="x", y="y",
    color="cluster",
    hover_data=["title"],
    title=f"Comparison of 6 Random Wikipedia Clusters: {', '.join(map(str,␣
↪random_clusters))}",
    width=1000, height=800
)
fig.show()
```

[ ]:
```
import plotly.express as px

# No noise (-1)
filtered_df = df[df['cluster'] != -1]

# downsample because otherwise I just wont load/ use way to much ram
filtered_df = filtered_df.sample(min(10000, len(filtered_df)))

fig = px.scatter(
    filtered_df,
    x="x", y="y",
    color="cluster",
    hover_data=["title"],
    title="All Wikipedia Clusters (Noise Removed, Sampled)",
    width=1000, height=800
```

```
)
fig.show()
```

[ ]:
```
import numpy as np
import plotly.express as px

# 100 random clusters
random_clusters = np.random.choice(df['cluster'].unique(), size=100,␣
↪replace=False)

filtered_df_100 = df[df['cluster'].isin(random_clusters)]

# downsample because otherwise I just wont load/ use way to much ram
filtered_df_100 = filtered_df_100.sample(min(10000, len(filtered_df_100)))

fig = px.scatter(
    filtered_df_100,
    x="x", y="y",
    color="cluster",
    hover_data=["title"],
    title="Comparison of 100 Random Wikipedia Clusters (sampled)",
    width=1000, height=800
)
fig.show()
```

[ ]:
```
import cupy as cp
from cuml.cluster import HDBSCAN
import matplotlib.pyplot as plt
from collections import Counter

labels_cpu = cp.asnumpy(labels)

num_clusters = len(set(labels_cpu)) - (1 if -1 in labels_cpu else 0)
print(f" Number of clusters found (excluding noise): {num_clusters}")

cluster_counts = Counter(labels_cpu)
sorted_clusters, sorted_sizes = zip(*sorted(cluster_counts.items(), key=lambda␣
↪x: x[0]))

plt.figure(figsize=(12, 6))
plt.bar([str(c) for c in sorted_clusters], sorted_sizes)
plt.title("Cluster Size Distribution")
plt.xlabel("Cluster ID (-1 = noise)")
plt.ylabel("Number of Items")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Number of clusters found (excluding noise): 1928
```

[ ]:
```python
#Convert labels from CuPy to NumPy
labels_cpu = cp.asnumpy(labels)

#(excluding noise)
num_clusters = len(set(labels_cpu)) - (1 if -1 in labels_cpu else 0)
print(f" Number of clusters found (excluding noise): {num_clusters}")

#items (excluding noise)
cluster_counts = Counter(label for label in labels_cpu if label != -1)
sorted_clusters, sorted_sizes = zip(*sorted(cluster_counts.items(), key=lambda␣
↪x: x[0]))

plt.figure(figsize=(12, 6))
plt.bar([str(c) for c in sorted_clusters], sorted_sizes)
plt.title("Cluster Size Distribution (Noise Removed)")
plt.xlabel("Cluster ID")
plt.ylabel("Number of Items")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


```
Number of clusters found (excluding noise): 1928
```

[6]:
```
df[df['cluster'] != -1][['cluster', 'x', 'y']].
↪to_csv("wikipedia_clusters_xy_no_noise.csv", index=False)
```

[7]:
```
df[['cluster', 'x', 'y']].to_csv("wikipedia_clusters_xy_with_noise.csv",␣
↪index=False)
```

4.2.1 Results HDBSCAN

As you might be able to see the clusters worked really well. It decided to make arround 17000 clusters. The clusters in of themselfs were semanticly good. The number of articles inside the cliusters are not evenly distributed in Wikipedia but that is not suprising. HDB had problems with noise (articles wehere it coudn't find 30 to cluster them together) The Graphs you see are projected down to 2 dminesions from 10 so the many of the information about the distance of clusters can not be displayed in that way.

Overall the HDBScan worked better, mainly because I didn't have to set the number of clusters myself

5 Supervised approach

Because the Supervidsed approach only works if I group some Wikipedia articles beforehand, like in the last Kaggle Project a supervised approach could then sort the rest into these groups. I implimented some dummy code to simualte that. Dummy because I don't want to label the wikipedia articles

[ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

```
np.random.seed(42)
embeddings_np = np.random.rand(1000, 100).astype('float32')
labels = np.random.choice(['Science', 'History', 'Music', 'Technology'],␣
↪size=1000)

# Step 2: Train/test split
X_train, X_test, y_train, y_test = train_test_split(embeddings_np, labels,␣
↪test_size=0.2, random_state=42)

# Step 3: Train classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Step 4: Predict and evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

6 Overall Discussion / Results

The Semantic clustering worked really well, both algorithms where able to deliver passable clusters of the topics of the wikipedia articles. Both of them had some downsides for example K-means I had to input the number of clusters mysef, whereas because of the min. articles HDBscan had for a cluster it had more problem with noise.

6.1 Hyperparamitisation

I changed the parameters by hand to find good and working parameters. Because of hardware limitations I was not able to Programmaticly find "the best" Hyperparameters

6.2 Usecase

Projects like these could be used with further refinement for axample to cluster and combine Company documentation. Many Times I have come across scattered Compay documentation without a red line to follow. Mit Semantic clustering I think some it it could be lessened. The alsorithm could precluster them and find a new headline to group them under to better find stuff when you search for it

6.3 Improvements for the future

6.3.1 What went wrong?

One of the main problems was GPU power and scope. The entire German wikipedia on a Laptop / household PC was way to large of a scope. Because of that I had to use only half of the german wikipedia halfway through because of comute times.

Also there could be better Hyperparemeterisation / reducing noise.

In the end I think I was to careful with the datacleaning, and I should have "eliminated" every article with under 4 Words further reducing noise.

6.3.2 Gerneral Improvements

Run this on a more powerful Hardware to compute the entirety. Use more complex embedding algorithms to get more semantic information

Extra Later I had acess to a A100 and I Clustered it without first reducing it to 30 dimensions insead working with 300. The parquet is available on Kaggle. But because of time constrains I don't have the time to rerun / rework my entire project

[2]:
```
#  Full GPU-based Wikipedia Embedding Clustering Pipeline

import time
import pandas as pd
import numpy as np
import cupy as cp
import cudf
from cuml import UMAP
from cuml.cluster import HDBSCAN

# Step 1: Load embeddings
print("  Loading data...")
df = pd.read_parquet("wikidump_half_title_embedding.parquet")

# Step 2: Convert embeddings to cuDF for GPU processing
print("  Converting embeddings to GPU format...")
start = time.time()
embeddings_np = np.vstack(df['embedding'].values).astype('float32')
embeddings_gpu_df = cudf.DataFrame(embeddings_np)
print(f"  Done in {time.time() - start:.2f} seconds.")

# Step 3: UMAP on GPU
print("  Running UMAP on GPU...")
start = time.time()
umap_model = UMAP(
    n_components=200,
    n_neighbors=30,
    metric='cosine',
    random_state=42,
    verbose=True
)
X_umap_gpu = umap_model.fit_transform(embeddings_gpu_df)
print(f" UMAP done in {time.time() - start:.2f} seconds.")

# Step 4: HDBSCAN on GPU
print("  Running HDBSCAN on GPU...")
```

```
start = time.time()
clusterer = HDBSCAN(
    min_cluster_size=30,
    metric='euclidean',  # UMAP changes the space, so Euclidean is fine here
    cluster_selection_method='eom'
)
labels_gpu = clusterer.fit_predict(X_umap_gpu)  # Output: cudf.Series
df['cluster'] = labels_gpu.to_pandas()
print(f" HDBSCAN done in {time.time() - start:.2f} seconds.")

# Step 5: Show summary
print(" Example cluster counts:")
print(df['cluster'].value_counts())

# Optional: Save result
# df.to_parquet("clustered_output_gpu.parquet")
```

```
 Loading data…
 Converting embeddings to GPU format…
 Done in 27.87 seconds.
 Running UMAP on GPU…
[2025-04-25 13:10:18.661] [CUML] [info] build_algo set to brute_force_knn
because random_state is given
[2025-04-25 13:10:18.903] [CUML] [debug] Computing KNN Graph
[2025-04-25 13:14:44.093] [CUML] [debug] Computing fuzzy simplicial set
 UMAP done in 363.49 seconds.
 Running HDBSCAN on GPU…
 HDBSCAN done in 176.04 seconds.
 Example cluster counts:
cluster
-1       622130
 39       29882
 2019     22220
 1996     21903
 851      21383
          …
 1526        30
 1799        30
 1255        30
 369         30
 1684        30
Name: count, Length: 2157, dtype: int64
```

[3]:
```
#  Step 1: Convert UMAP dimensions to a DataFrame
umap_columns = [f'umap_{i}' for i in range(X_umap_gpu.shape[1])]
umap_df = X_umap_gpu.to_pandas()
umap_df.columns = umap_columns
```

```python
#  Step 2: Convert cluster labels to pandas
labels = labels_gpu.to_pandas()

#  Step 3: Merge everything
df_umap = pd.concat([df.reset_index(drop=True), umap_df], axis=1)
df_umap['cluster'] = labels

#  Step 4: Export
df_umap.to_parquet("clustered_embeddings_with_umap.parquet", index=False)
# Or if you prefer CSV:
# df_umap.to_csv("clustered_embeddings_with_umap.csv", index=False)

print(" UMAP dimensions and clusters saved!")
```

```
 UMAP dimensions and clusters saved!
```

[4]:
```python
print(df_umap.head())
```

```
                 title                                          embedding  \
0          Peter Giger  [0.017257495, 0.0020586113, -0.035239954, -0.0…
1        S-25 (U-Boot)  [0.01616242, -0.0037445973, -0.033856593, -0.0…
2  Gewöhnliche Goldrute  [0.016481167, 0.018800229, -0.014906349, -0.06…
3           Kerckhoffs  [0.009618139, 0.029865338, -0.023966001, -0.05…
4      Scipione (Maler)  [0.0044155587, -0.014298238, -0.011290375, -0…

   cluster    umap_0    umap_1    umap_2    umap_3    umap_4    umap_5  \
0       -1 -0.198943 -0.154436 -0.154515 -0.094365 -0.447018 -0.589359
1     1716 -0.660520 -0.245930 -0.227188 -0.187867 -0.432676 -0.276781
2       85  0.011758 -0.131212 -0.268349 -0.007226 -0.042724 -0.448263
3     1552 -0.026503  0.032446  0.471517  0.016047  0.217901  0.779350
4     1407 -0.060193 -0.148629 -0.204621 -0.194108 -0.122166 -0.204509

     umap_6  …  umap_190  umap_191  umap_192  umap_193  umap_194  umap_195  \
0 -0.051906  … -0.363674 -0.142809 -0.321787  0.158234 -0.779624 -0.355350
1  0.055142  …  0.186186 -0.117536 -0.249324  0.014158 -0.222038 -0.141085
2 -0.105216  … -0.401830  0.332285 -1.108608 -0.015889  0.098654 -0.130384
3 -0.347778  … -0.134467  0.013834  0.089489  0.181625 -0.039921 -0.105259
4 -0.068975  …  0.040904 -0.074720 -0.413085 -0.369158 -0.409154 -0.099329

   umap_196  umap_197  umap_198  umap_199
0 -1.015008 -0.104247 -0.189773 -0.258785
1  0.066126 -0.133408 -0.198235 -0.193342
2  0.685607 -0.048760 -0.097721  0.092986
3 -0.099517 -0.302877  0.201938  0.148233
4 -0.223391 -0.086782 -0.275490 -0.199657

[5 rows x 203 columns]
```

```
[ ]:
```

6.3.3 Tools I Used

- I used ChatGPT to impliment timings into my code, sometimes it that changed a few parts of the code when I only wanted to implement the abilyty to estimate the runtime. (Minutes or Hours or days)
- I also used Huggingface Leaderbord for semantic embeddings, the Wikidums, and Chatgpt to help be find the right embedding algorithms / clustering types.
- Chagpt to impliment visual Print statements (and with that sometimes comments becuase GPT cant help itself)